Internet of Things
2016/2017

CoAP

John Carpenter, 1982

Johan Lukkien

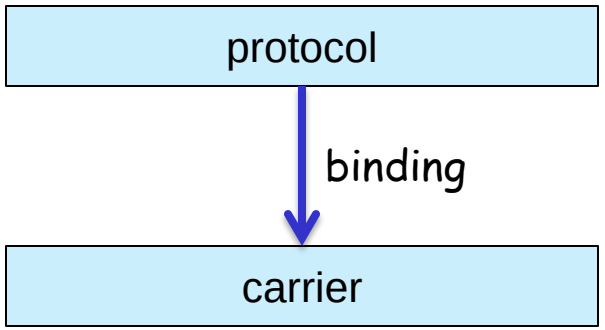
Johan J. Lukkien, j.j.lukkien@tue.nl
TU/e Informatica, System Architecture and Networking

Guiding questions

- What makes protocols ‘low resource’, and what are examples?
- How to assess protocols quickly?
- How does CoAP work in particular and how does it help low resource devices?

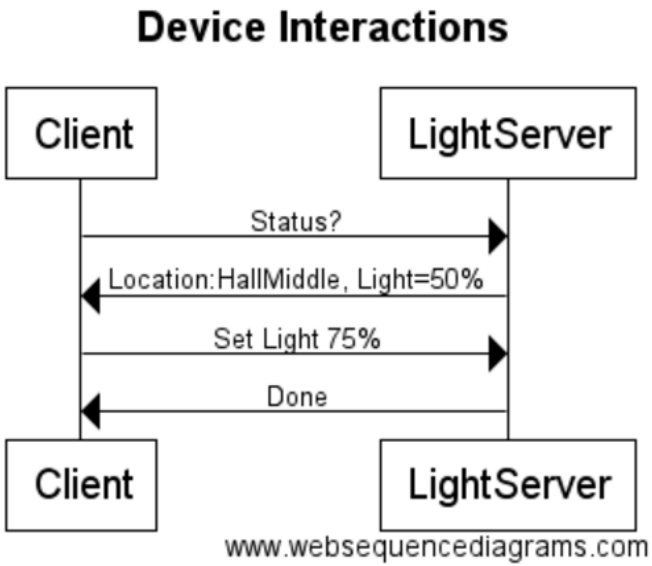
Framework for discussing protocols

- Purpose of the protocol
  - the problems it solves
  - the context it operates in, the place in the stack
- Parties that use it
- Functionality, typical behavior
- Packet format
- Carriers
- Binding to carriers
- Utility for IoT



Application level interactions

- Applications consists of coordinated request/reply interactions, e.g.,
  - functional scenarios
  - management scenarios
  - data collection by P&S
- Notice that the IoT device is the *server*
  - must be findable, accessible
  - must be protected against misuse
- CoAP is an application level IoT protocol, standardized by the IETF

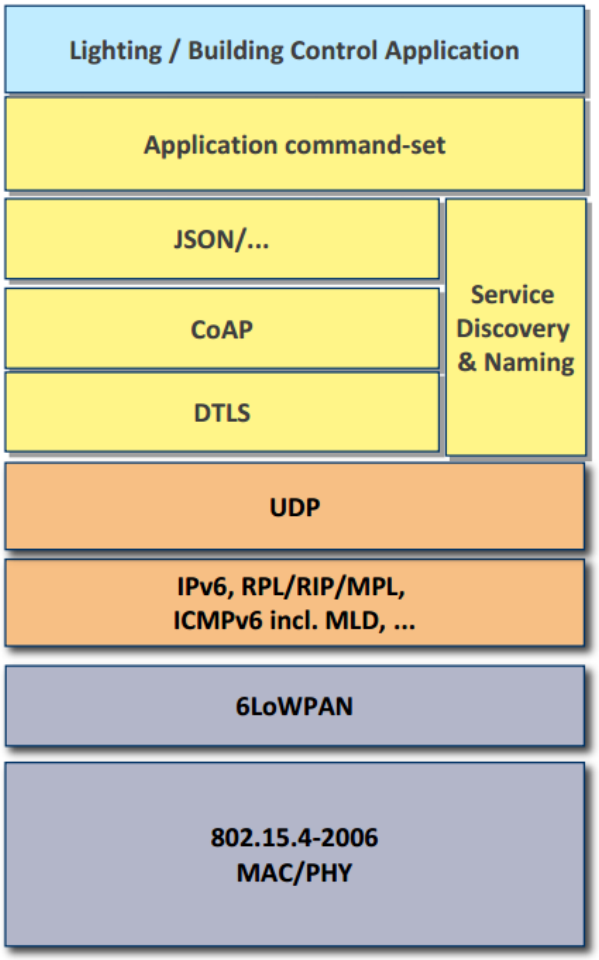


CoAP – problems it solves (requirements)

- Support for Internet REST operation
  - (remote) resource manipulation as the basic concept; use of URIs
  - caching
  - straightforward mapping / translation to HTTP
- Enable machine to machine operation
  - resource discovery
  - publish / subscribe / notify
  - multicast
- Reduce inefficiencies in REST operation
  - the encoding scheme (text) and message size
  - the carrier: TCP
- Use limited resources
  - small footprint
  - constrained networks
  - support sleeping nodes – proxy-ing

CoAP context, place in the stack

- CoAP users
  - Users of simple web services
    - CoAP implements a web services protocol
  - Other CoAP devices – machines
  - Management servers
    - e.g. using LWM2M, accessing and managing a constrained device through CoAP
- CoAP is an application layer protocol, and serves applications
  - it is not an application itself
- Binding of CoAP has been defined as UDP payload
  - port 5683, 5684 for secure CoAP on top of DTLS
- Binding for SMS has been defined as well



Courtesy of Dee Denteneer

Functionality CoAP provides

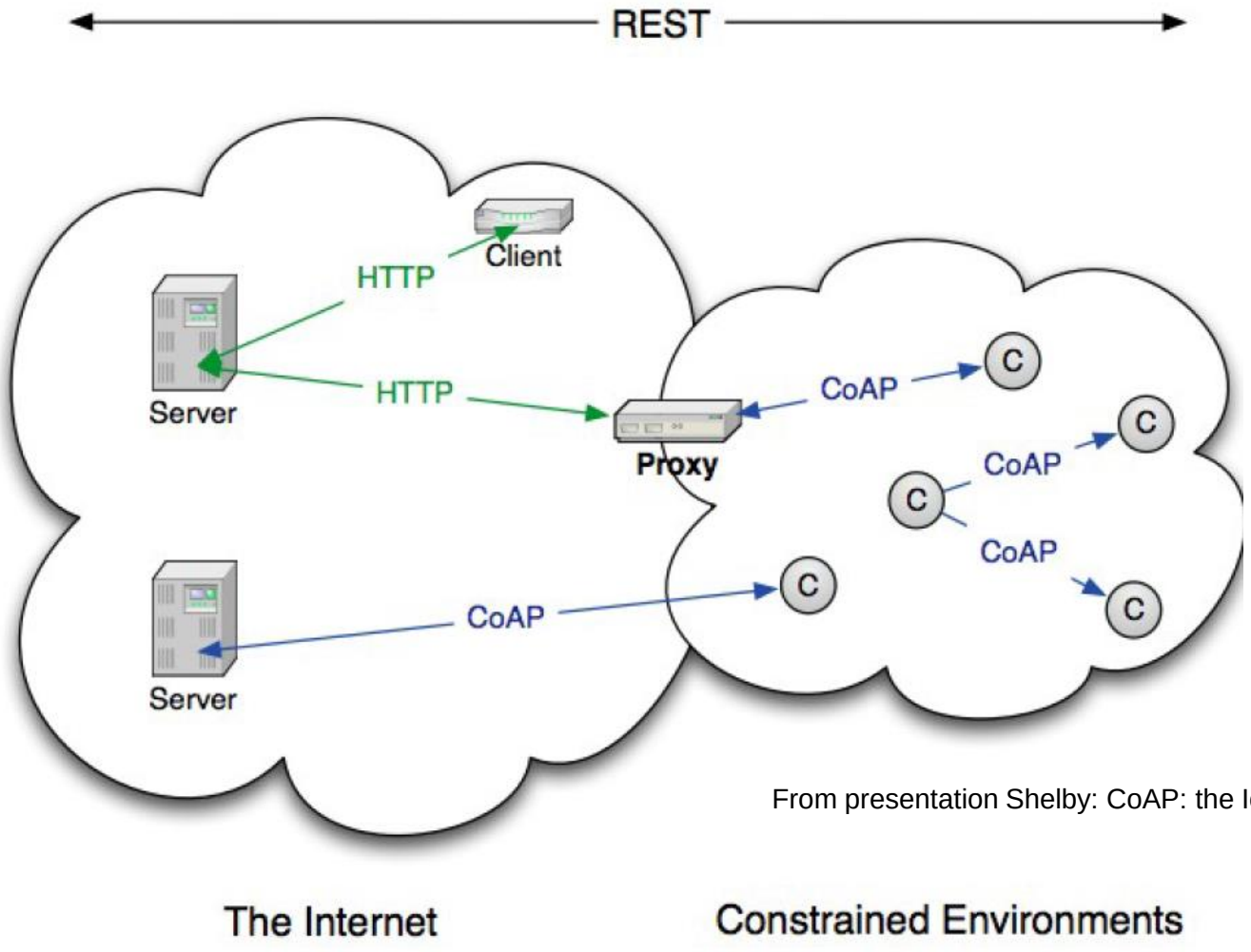
- CoAP-controlled transaction reliability
  - transaction: a set of message exchanges that implement a REST call
- RESTful interface

- Caching of responses, and caching control
- Proxy-ing

- Block oriented transfer
- Resource discovery

- Characteristic: short messages

End-to-end REST operation



From presentation Shelby: CoAP: the IoT protocol

CoAP messages

- Reliability under application control: transaction semantics moved from transport protocol to CoAP
  - transaction semantics: confirmation of messaging, reliability of the message exchange

- Message *types* determine message role as part of the transaction
- Message *codes* determine the further purpose of the message
  - request or response
- Message *id* specifies the transaction
- Message *token* specifies a concept of topic: set of messages belonging together
- Message *options* specify parameters and handling mechanisms

CoAP transactions

	CON	NON	ACK	RST
Request	X	X	-	-
Response	X	X	X	-
Empty	*	-	X	X

ping

Message types

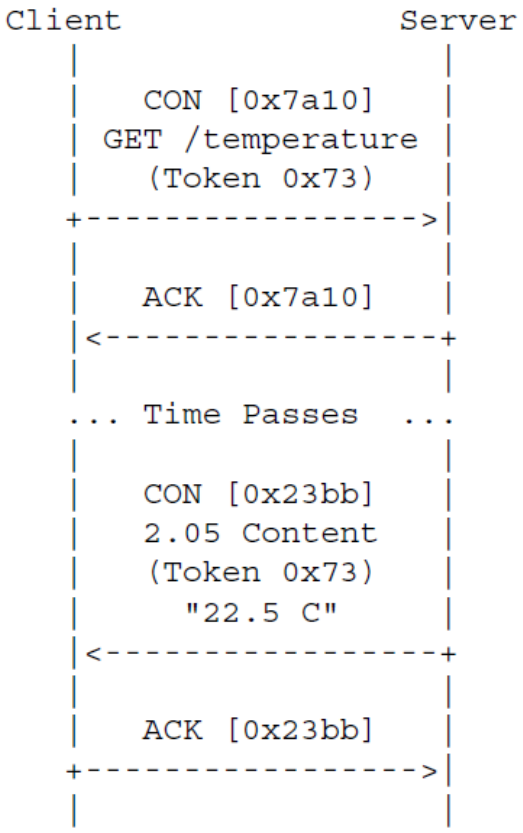
- CON
  - confirmable, respond with ACK
- NON
  - non-confirmable
- ACK
  - acknowledge CON, piggy-back with response
- RST
  - deny (reset) interaction
- ping
  - empty CON, responded with empty ACK

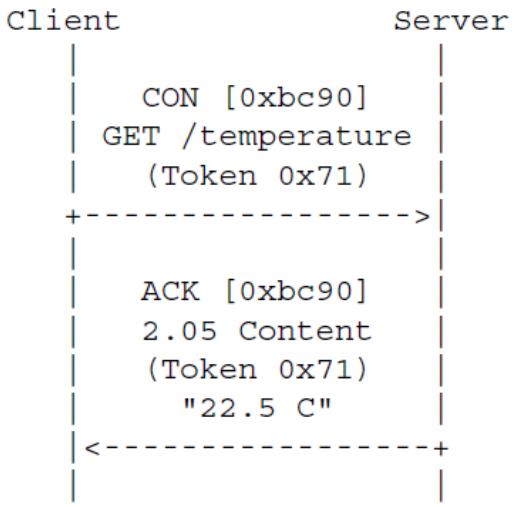
Contents

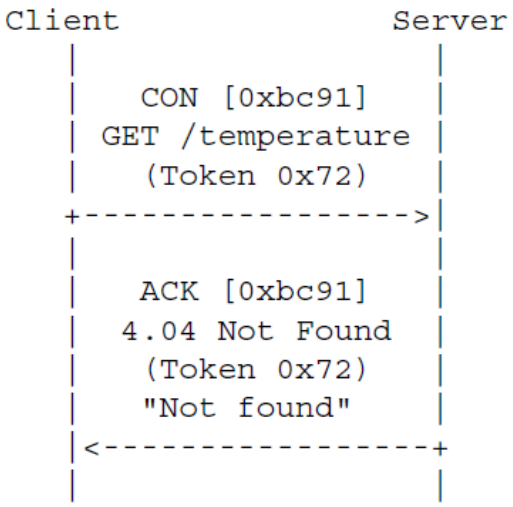
- Code
  - Request method or Response
  - method: GET, PUT, POST, DELETE
- Message ID (transaction ID)
- Numbered options, e.g.,
  - specify URI
  - specify media type
- Optional token
  - ‘interaction ID’, ‘topic’
- (Payload)

Acknowledgement versus Response

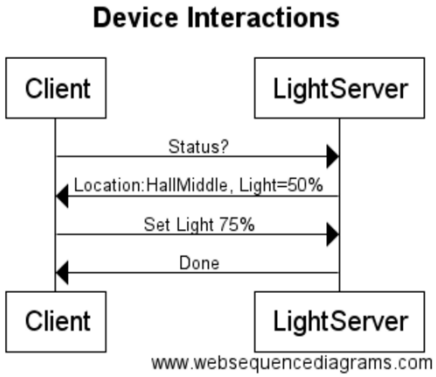
- An ACK confirms receipt of a CON
- A response gives the result of a request
  - transmitted using CON / ACK again
  - may piggy-back on the first ACK







Requests and Responses



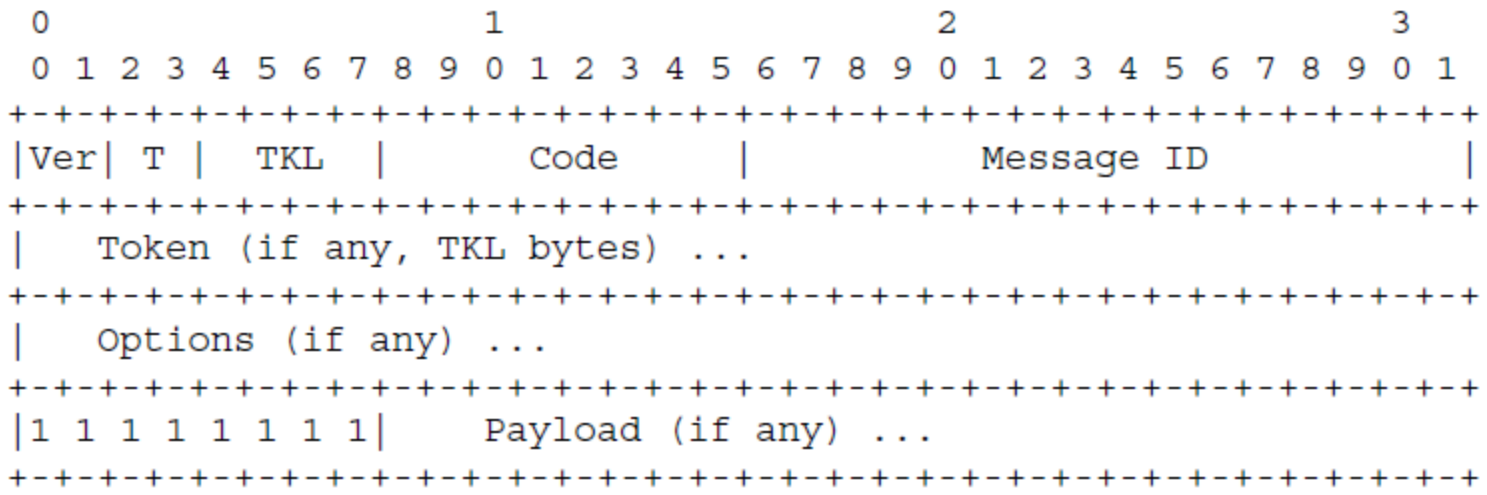
Request methods

- GET URI
  - retrieve (resource identified by) URI
- PUT URI [parameter]
  - update URI
- DELETE URI
  - delete URI
- POST URI [parameter]
  - perform operation and create new resource under URI
- GET is safe (no side effects)
- GET, PUT and DELETE are (have to be) idempotent

Responses

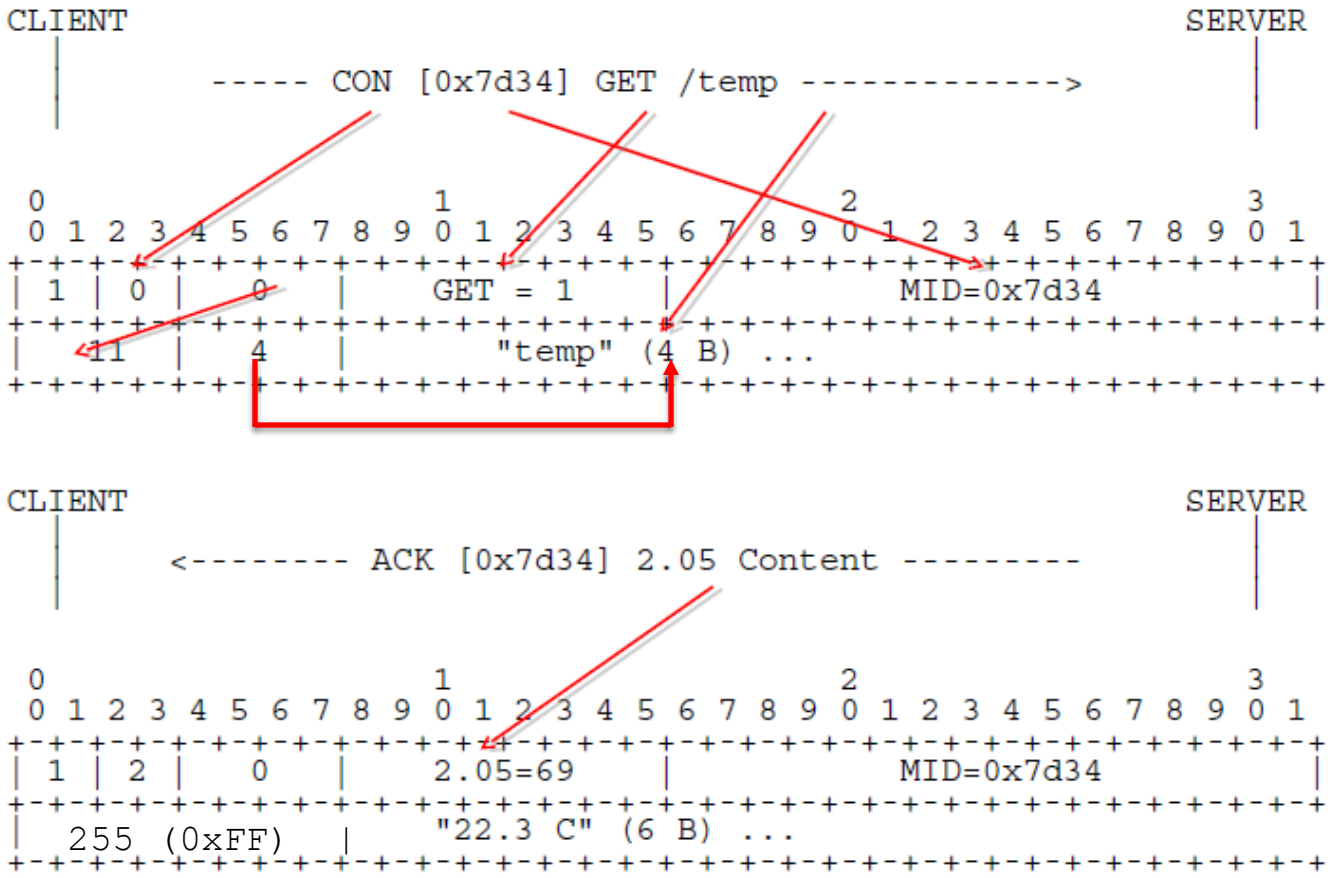
- Class 2 (success)
  - 2.01: created, 2.02: deleted, 2.03: valid, 2.04: changed, 2.05: content
- Class 4 (client error)
  - 4.00: bad request, 4.01: unauthorized, 4.02: bad option, 4.03: forbidden, 4.04: not found, 4.05: method not allowed, 4.06: not acceptable, 4.12: precondition failed
- Class 5 (server error)
  - 5.00: internal server error, 5.01: not implemented, 5.02: bad gateway, 5.03: service unavailable, 5.04: gateway timeout, 5.05: proxying not supported

The packet format



- Ver(sion) – 1
- T(ype) – CON / ACK / NON / RST
- T(o)K(en)L(ength)
- (Request or Response) Code: GET, PUT, POST, DELETE + responses
- 0xFF: payload marker

Mapping



(9 bytes)

From presentation Shelby: CoAP: the IoT protocol

Options

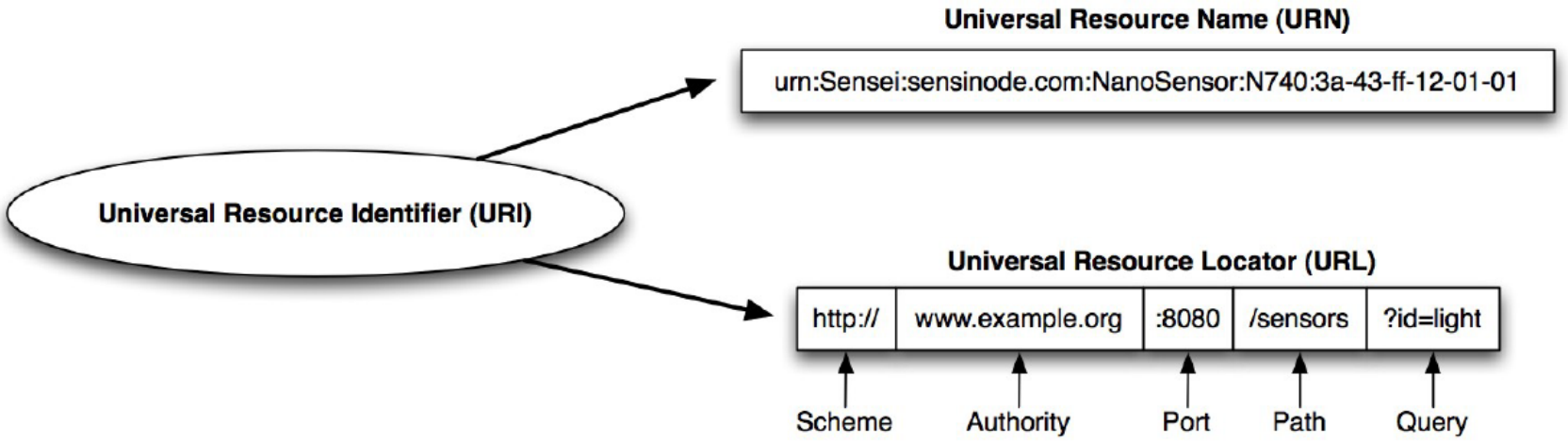
- Examples:
  - specify different components of the URI in the request

No.	C	U	N	R	Name	Format	Length	Default
1	x			x	If-Match	opaque	0-8	(none)
3	x	x	-		Uri-Host	string	1-255	(see below)
4				x	ETag	opaque	1-8	(none)
5	x				If-None-Match	empty	0	(none)
7	x	x	-		Uri-Port	uint	0-2	(see below)
8				x	Location-Path	string	0-255	(none)
11	x	x	-	x	Uri-Path	string	0-255	(none)
12					Content-Format	uint	0-2	(none)
14		x	-		Max-Age	uint	0-4	60
15	x	x	-	x	Uri-Query	string	0-255	(none)
17	x				Accept	uint	0-2	(none)
20				x	Location-Query	string	0-255	(none)
35	x	x	-		Proxy-Uri	string	1-1034	(none)
39	x	x	-		Proxy-Scheme	string	1-255	(none)
60			x		Size1	uint	0-4	(none)

C=Critical, U=Unsafe, N=NoCacheKey, R=Repeatable

Naming on the Internet

- Name: identity
- Locator: address



From presentation Shelby: CoAP: the IoT protocol

coap://example.tue.nl:5683/sensors/light1

coaps://example.tue.nl:5683/sensors/light1

Options

- Examples:
  - specify different components of the URI in the request
  - request a proxy to answer (cached, or forward) a URI

No.	C	U	N	R	Name	Format	Length	Default
1	x			x	If-Match	opaque	0-8	(none)
3	x	x	-		Uri-Host	string	1-255	(see below)
4				x	ETag	opaque	1-8	(none)
5	x				If-None-Match	empty	0	(none)
7	x	x	-		Uri-Port	uint	0-2	(see below)
8				x	Location-Path	string	0-255	(none)
11	x	x	-	x	Uri-Path	string	0-255	(none)
12					Content-Format	uint	0-2	(none)
14		x	-		Max-Age	uint	0-4	60
15	x	x	-	x	Uri-Query	string	0-255	(none)
17	x				Accept	uint	0-2	(none)
20				x	Location-Query	string	0-255	(none)
35	x	x	-		Proxy-Uri	string	1-1034	(none)
39	x	x	-		Proxy-Scheme	string	1-255	(none)
60			x		Size1	uint	0-4	(none)

C=Critical, U=Unsafe, N=NoCacheKey, R=Repeatable

Options

- Examples:
  - specify different components of the URI in the request
  - request a proxy to answer (cached, or forward) a URI
  - specify formatting info of content (payload) and of acceptable formats
    - there is a table of known formats
    - ***content-format*** 50: the payload is encoded in application/json
    - ***accept*** 0: I accept (prefer) text/plain
  - give ***Max-Age***, used in caching responses (60s default)
    - "you may cache this response for Max-Age seconds"
    - must be original and fresh!
  - ***Location-path****:* give the location of a newly created resource
  - ***Etag****:* validate cached value
    - "is this value still valid?"

Options have properties

- Critical / elective
  - whether or not the receiver must understand the option
  - this determines response to errors in the options
    - report or ignore
- (Un)safe to forward
  - determines proxy forwarding in case of error in options, or message not understood
  - is it safe to forward even if I don't understand it?
- NoCacheKey
  - Cache Key: a key defining a query and assigned to the response
    - admits looking up a response based on a received query
  - A 'NoCacheKey' option is not considered when matching a request to see if a cached response is possible
  - for considering an option as CacheKey it must be safe to forward
- Repeatable
  - option may occur more than once

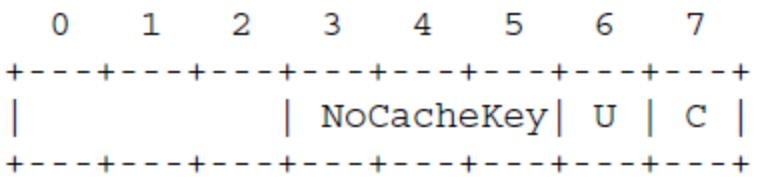
Options and their properties

No.	C	U	N	R	Name	Format	Length	Default
1	x			x	If-Match	opaque	0-8	(none)
3	x	x	-		Uri-Host	string	1-255	(see below)
4				x	ETag	opaque	1-8	(none)
5	x				If-None-Match	empty	0	(none)
7	x	x	-		Uri-Port	uint	0-2	(see below)
8				x	Location-Path	string	0-255	(none)
11	x	x	-	x	Uri-Path	string	0-255	(none)
12					Content-Format	uint	0-2	(none)
14		x	-		Max-Age	uint	0-4	60
15	x	x	-	x	Uri-Query	string	0-255	(none)
17	x				Accept	uint	0-2	(none)
20				x	Location-Query	string	0-255	(none)
35	x	x	-		Proxy-Uri	string	1-1034	(none)
39	x	x	-		Proxy-Scheme	string	1-255	(none)
60			x		Size1	uint	0-4	(none)

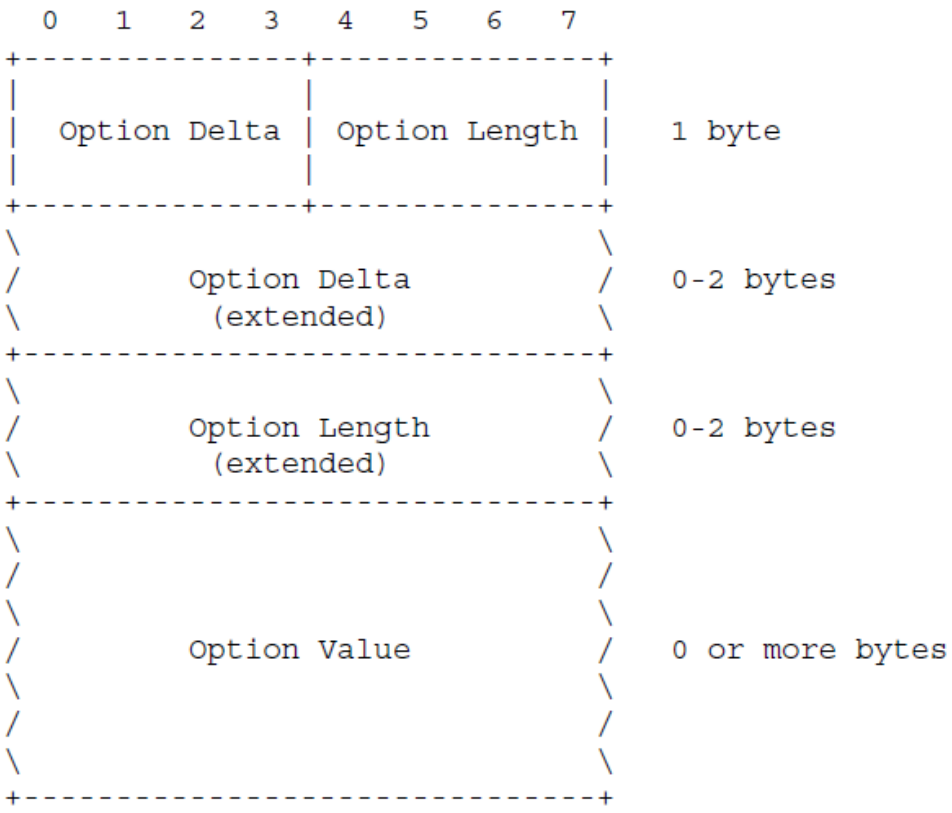
- x: true
- space: false
- -: not applicable

C=Critical, U=Unsafe, N=NoCacheKey, R=Repeatable

Option properties encoded in option number (NoCacheKey=111)



Options encoding within a message



- Series of options
- Option Delta: offset from previous option number
  - hence, options listed in increasing order
  - the current option is the sum of the delta's until now
  - 13,14 reserved for extending
    - 1 or 2 bytes respectively
  - 15: payload marker
    - length = 15 as well
- Option Length: of the option value that follows it
  - 13,14 extending again

The OBSERVE option (RFC 7641)

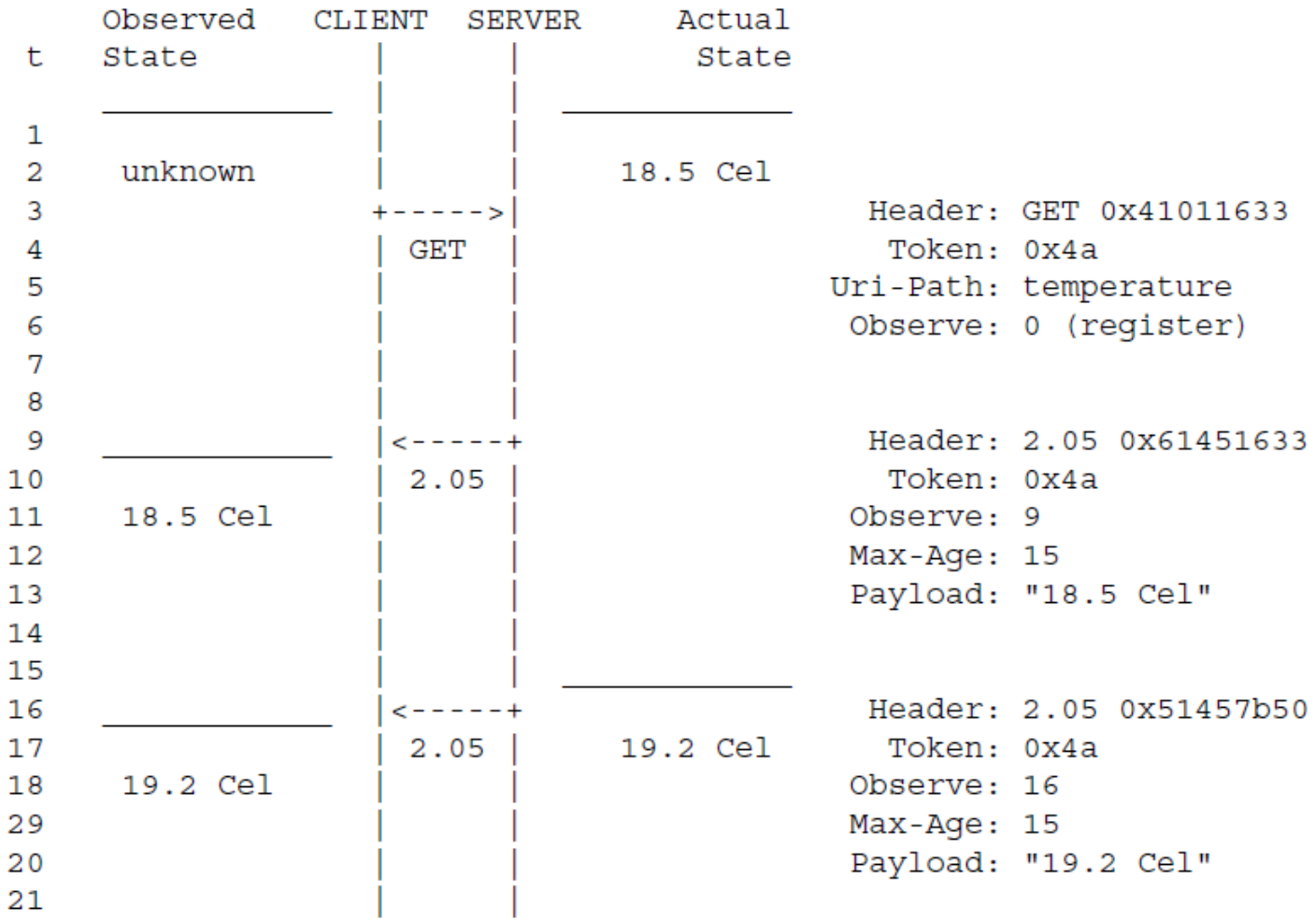
No.	C	U	N	R	Name	Format	Length	Default
6		x	-		Observe	uint	0-3 B	(none)

C=Critical, U=Unsafe, N=No-Cache-Key, R=Repeatable

- In GET:
  - obtain current resource representation
  - add/remove to list of observers (value 0/1 respectively)
- As a reponse:
  - interpret response as notification
  - value is sequence number

- Moves from pull to push based updates

Example


Observed State
CLIENT
SERVER
Actual State
t
unknown
18.5 Cel
GET
2.05
18.5 Cel
2.05
19.2 Cel
19.2 Cel
Header: GET 0x41011633
Token: 0x4a
Uri-Path: temperature
Observe: 0 (register)
Header: 2.05 0x61451633
Token: 0x4a
Observe: 9
Max-Age: 15
Payload: "18.5 Cel"
Header: 2.05 0x51457b50
Token: 0x4a
Observe: 16
Max-Age: 15
Payload: "19.2 Cel"

Multiplexing mechanisms

- On a node there may be different servers active, or different information streams
  - distinct resources with independent access mechanisms
  - serving listeners to different types of events

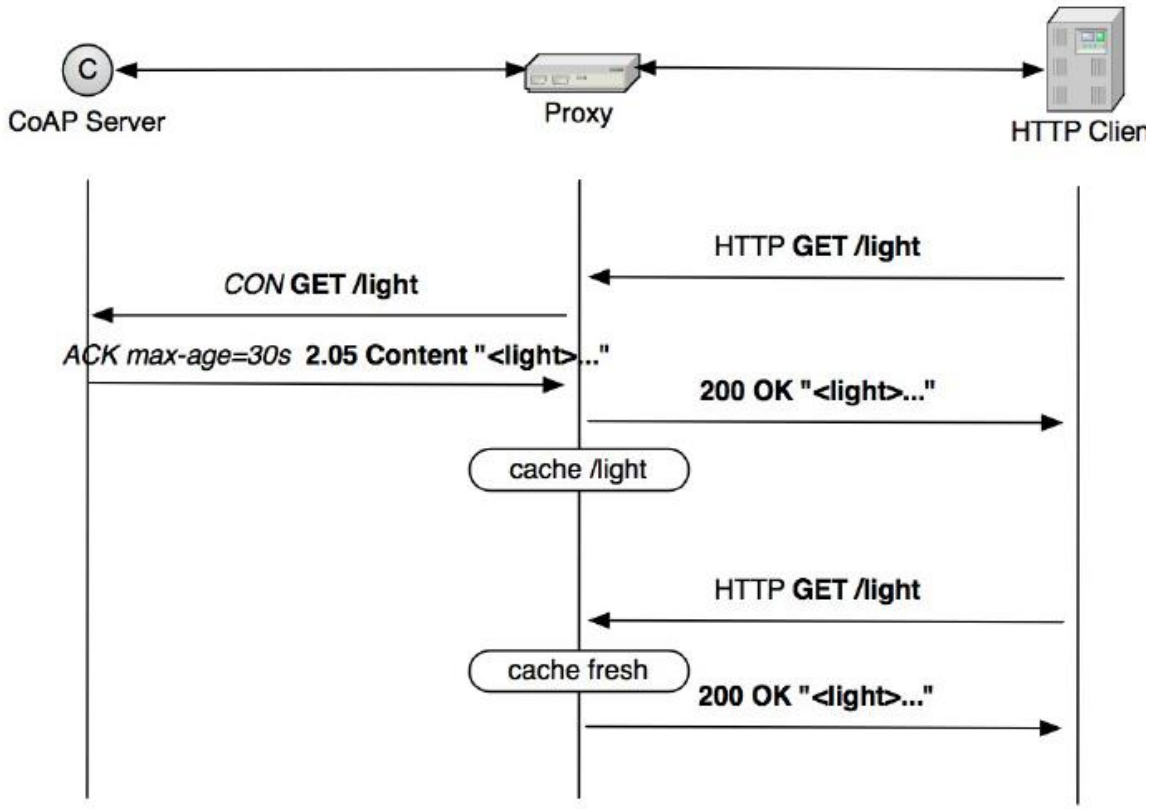
- Mechanisms to discriminate different traffic streams
  - multiple ports
    - requires multiple servers, application dependent port choice
    - CoAP(S) is associated to a port like HTTP(S)
  - resource tree in URI
    - requires dispatching to handler of the resource
  - message token
    - dispatching
    - traffic type classification – grouping exchanges – handled by single server

Caching

- Motivation for caching:
  - reduce network load
  - reduce load on nodes
- Cache-ability derives from response code
  - the response defines whether it is cache-able (there is a table)
  - a cache key (for matching) is determined by
    - request
    - options in request that are selected as cache key
- Max-Age option in response determines validity
- Validation through the Etag option

Proxy-ing

- Forward proxy
  - act as a client on behalf of another client
  - example: HTTP proxy
- Reverse proxy
  - act as a server on behalf of a server (a low node) that is sleeping or of too low resource
- Proxies may perform caching and may change protocol



Configuration parameters

name	default value
ACK_TIMEOUT	2 seconds
ACK_RANDOM_FACTOR	1.5
MAX_RETRANSMIT	4
NSTART	1
DEFAULT_LEISURE	5 seconds
PROBING_RATE	1 byte/second

name	default value
MAX_TRANSMIT_SPAN	45 s
MAX_TRANSMIT_WAIT	93 s
MAX_LATENCY	100 s
PROCESSING_DELAY	2 s
MAX_RTT	202 s
EXCHANGE_LIFETIME	247 s
NON_LIFETIME	145 s

Concluding

- CoAP supports HTTP-style RESTful applications
- CoAP reduces TCP carrier overhead and brings it under control of the application
- CoAP reduces the data-size overhead of HTTP/TCP significantly
  - typical GET and response just a few bytes in size
- CoAP support a variety of in-network behaviors/mechanisms that improve performance of low-resource devices
  - proxy, P&S, caching
- CoAP deals effectively with peculiarities of low resource nodes

Guiding questions

- What makes protocols ‘low resource’?
- How to assess protocols quickly?
- How does CoAP work in particular and how does it help low resource devices?

Framework for discussing protocols

- Purpose of the protocol
  - the problems it solves
  - the context it operates in, the place in the stack
- Parties that use it
- Functionality, typical behavior
- Packet format
- Carriers
- Binding to carriers
- Utility for IoT